Adequate Shear Measurements Using Gaussian Mixtures (and Bayesian stuff)

Erin Sheldon
Brookhaven National Laboratory

Gaussian Mixture

$$G = \frac{p}{2\pi\sqrt{|\mathrm{M}|}} \exp\left(-\frac{1}{2}\mathrm{X}^T\mathrm{M}^{-1}\mathrm{X}\right)$$

$$I = \sum_{i=1}^{N_g} p_i G_i$$

$$\begin{aligned} I_o &= \sum_{i=1}^{N_g} p_i \sum_{j=1}^{N_P} G_i * P_j \\ &= \sum_{i=1}^{N_g} p_i \sum_{j=1}^{N_P} \frac{1}{2\pi\sqrt{|\mathrm{M}_o|}} \exp\left(-\frac{1}{2}\mathrm{X}^T\mathrm{M}_o^{-1}\mathrm{X}\right) \\ M_o &= M + M_P \end{aligned}$$

How Many, Which Way?

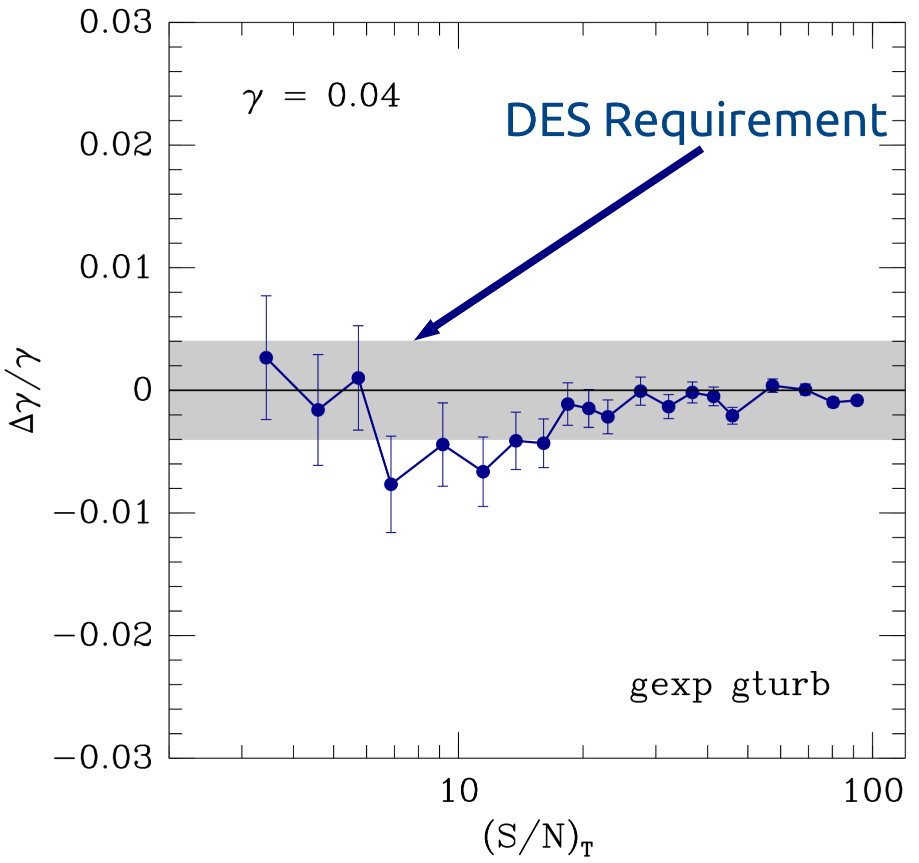
- Three Gaussians seems to be enough when the galaxy is comparable in size of the PSF. One is a delta function.
- More Gaussians are needed for galaxies much larger than the psf.
- For efficiency just throw out the really big galaxies (there are few), or let the algorithm adapt (T test?)
- Hogg et al. 2012 Independently find that, for flux measurements, Gaussians are a good representation
- For galaxies choose Co-centric, co-elliptical to avoid degeneracies. PSF can be more free.

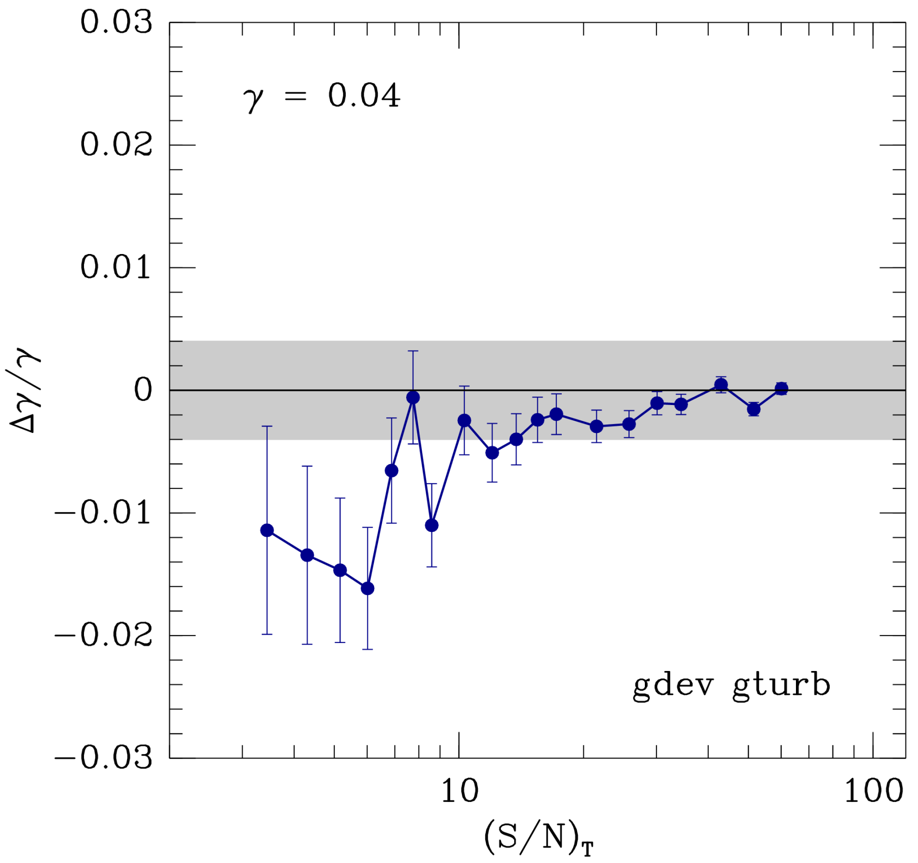
Bayesian Methods

- Apply prior on shapes, necessarily zero-centered
- Use expectation value of the ellipticity (not the maximum likelihood).
- Apply a correction for noise and the zero-centeredness of prior (Miller et al. 2007)
- Requires measurement of the full likelihood.

$$\frac{\partial \langle e \rangle}{\partial g} \sim 1 - \left[\frac{\int (\langle e \rangle - e) L(e) \frac{\partial P}{\partial e} \mathrm{d}e}{\int P(e) L(e) \mathrm{d}e} \right]$$

Simulations





Cut at S/N_{size} > 20

S/N of the size seems universal
(for a universe with 2 galaxies)

- Unlike detection S/N or shape error, the S/N of the size measurement seems to be a good indicator of badness for both Dev. And Exp galaxies.
- (S/N)_{size} > 20 meets the DES spec in sims.
- Algorithm is super slow because of the full likelihood evaluation, even with a fast exp()
- Get factor of ~100 speedup with GPUs...

Extra Slides

Computationally Intensive

- Corrections for priors require measuring the full likelihood surface. Need to measure “tails” of the distributions precisely.
- Using a fast exponential function (5x faster than C stdlib) and Monte Carlo Markov Chain (MCMC) to efficiently explore the surface.
- Takes ~2 seconds per galaxy on a CPU (2.7GHz Intel)
- For DES we must process ~5 billion galaxy images, thus ~3 million cpu-hours for each run.
- Want many runs.
- We don't have much money, so need to speed this up.

GPU Implementation

- Implemented in OpenCL and C
- Bunch of boiler-plate, pack the data, send it to the GPU, run a “kernel” on every element of the array.
- Standard C code except for the opencl kernel, which is compiled at runtime.
- The kernel is simply the evaluation of a sum of Gaussians

Pixel Parallelization

- Evaluate all pixels at once in parallel.
- Speedup comes from parallelization over pixels, GPU cores themselves are not faster than CPU cores.
- Factor of > 10 speedup, depending on number of gaussians evaluated (greater for more gaussians)
- That's great, but we also have > 10 CPU cores on a system these days so overall that isn't an enormous speedup.
- We are not even close to fully utilizing the GPU for a 25x25 pixel image

Parallelizing “Walkers”

- Remember we evaluate ~20 separate “walkers” through the likelihood space at each step in the MCMC chain
- We evaluate the models for all walkers in parallel as well.
- In principle a factor of 20 speedup.
- In practice 10-15
- memory bandwidth (144GB/s) and exhaustion of cores (448).

Packing the Data

- It is slow to move data back and forth from the GPU
- Upload the image once, then perform tens of thousands of likelihood evaluations. (Pad to multiple of 32 bytes).
- For each step in the MCMC chain, upload all parameters at once. This means packing the parameters for all gaussians at all points in the parameter space (remember we use 20 parallel “walkers”) into a single array (padded).
- Evaluate the model for each walker onto a single big shared array holding all rendered images.
- Perform reductions sum((image-model)^2) for each of the 20 walkers and return just 20 numbers: the likelihoods for each walker
- Re-use the arrays for each step in MCMC chain

Optimizations

- Avoid passing data back and forth to GPU: generate on GPU if possible
  - But there is a trade-off between evaluating ahead of time and sending data to GPU vs evaluating on GPU
- Constants are fast: any fixed quantity used by all threads
- Keep the kernel as straightforward as possible
  - Avoid branches
  - Avoid loops: I unroll all loops, pre-generate kernels with a script (I hear loops *can* be OK in some cases)
- Keep the actual number of program runs on GPU low: huge overhead. Better to pack the data and evaluate separately. Faster than parallel program runs as well.